A Sierra Monitor Company

Driver Manual
(Supplement to the FieldServer Instruction Manual)

FS-8700-01 Modbus RTU
&
FS-8700-08 Modbus ASCII

There are several similarities between these two drivers and we have incorporated them into the same manual to ensure that our information stays current. Although both drivers are referenced in this manual, they are different drivers and need to be ordered separately."

APPLICABILITY & EFFECTIVITY

Effective for all systems manufactured after May 1, 2001

Instruction Manual Part Number: T28700-01
Rev. A.

Table of Contents

1. Modbus RTU/ Modbus ASCII Description

The Modbus RTU and Modbus ASCII drivers allow the FieldServer to transfer data to and from devices over either RS-232 or RS-485 using Modbus RTU or Modbus ASCII protocol respectively. The driver was developed for Modbus Application Protocol Specification V1.1a" from Modbus-IDA. The specification can be found at www.modbus.org. The FieldServer can emulate either a Server or Client.

The information that follows describes how to expand upon the factory defaults provided in the configuration files included with the FieldServer.

There are various register mapping models being followed by various vendors
To cover all these models FieldServer uses the following three Models

- **Modicon_5digit** – Use this format where addresses are defined in 0xxxx, 1xxxx, 3xxxx or 4xxxx format. A maximum of 9999 registers can be mapped of each type. This is FieldServer driver's default format.
- **ADU** –Application Data Unit address. Use this format where addresses of each type are defined in the range 1-65536
- **PDU** –Protocol Data unit address. Use this format where addresses of each type are defined in the range 0-65535.

The key difference between ADU and PDU is for example if Address_Type is ADU and address is 1, the driver will poll for register 0. If Address_Type is PDU, the driver will poll for address 1.

Note 1: If vendor document shows addresses in extended Modicon (i.e. 6 digit) format like 4xxxxx then consider these addresses as xxxxx (omit the first digit) and use either ADU or PDU

Note 2: The purpose of providing 3 different ways of addressing the Modbus registers is to allow the user to choose the addressing system most compatible with the address list being used. At the protocol level, the same protocol specification is used for all three with the exception of the limited address range for Modicon_5digit.

2. Driver Scope of Supply

2.1. Supplied by FieldServer Technologies for this driver

FIELDSERVER TECHNOLOGIES PART #	DESCRIPTION
FS-8915-10	UTP cable (7 foot) for RS-232 use
FS-8917-02	RJ45 to DB9F connector adapter
FS-8917-01	RJ45 to DB25M connection adapter
FS-8917-21	RS-485 connection adapter
FS-8700-01 (T28700-01)	Driver Manual

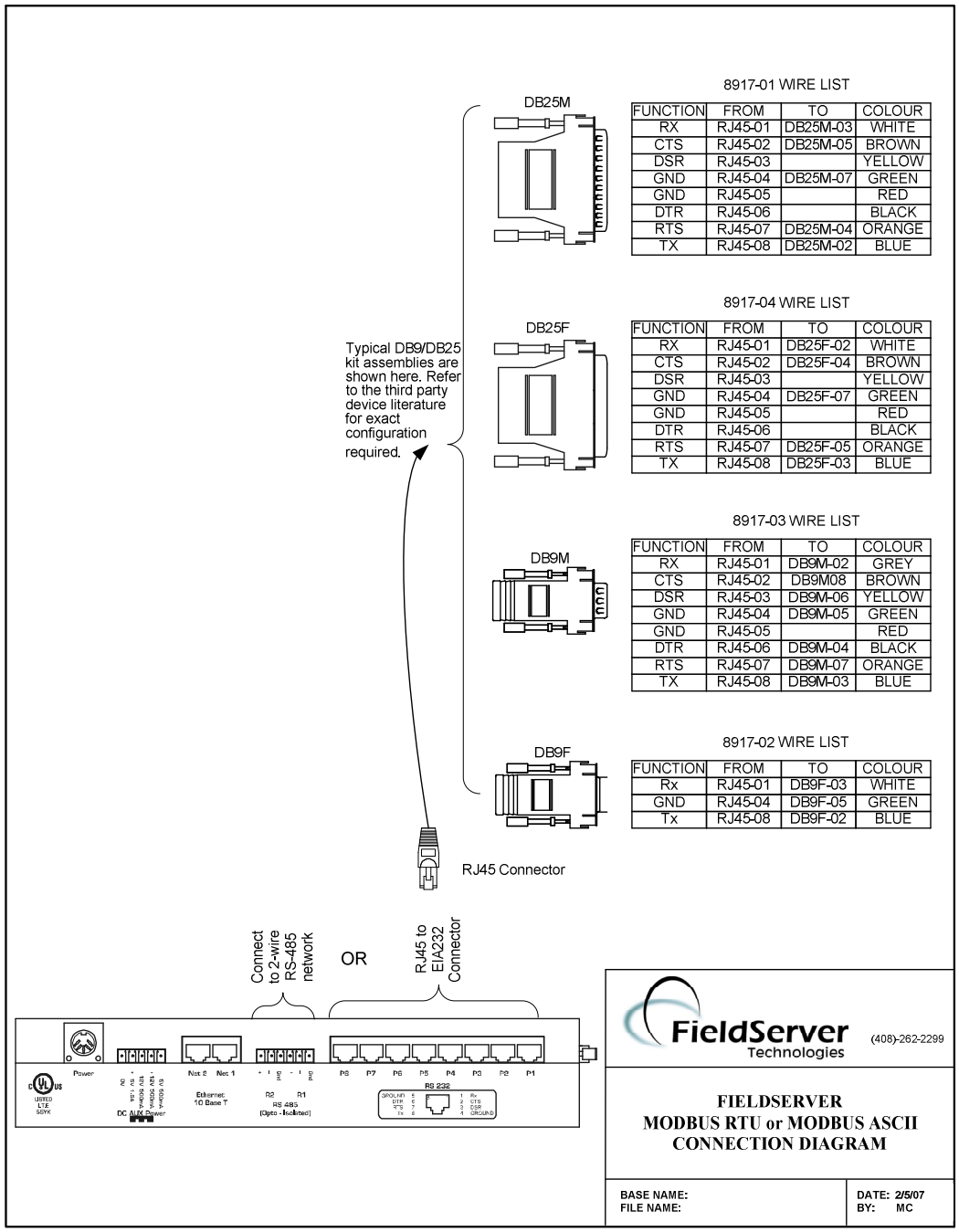
2.2. Provided by Supplier of 3rd Party Equipment

PART #	DESCRIPTION
	Modbus RTU or Modbus ASCII device

3. Hardware Connections

It is possible to connect a Modbus RTU or Modbus ASCII device to any of the existing serial ports on the FieldServer[1]. These ports simply need to be configured for the appropriate driver in the configuration file.

Configure the Modbus RTU or Modbus ASCII device according to manufacturer's instructions.



8917-01 WIRE LIST

FUNCTION	FROM	TO	COLOUR
RX	RJ45-01	DB25M-03	WHITE
CTS	RJ45-02	DB25M-05	BROWN
DSR	RJ45-03		YELLOW
GND	RJ45-04	DB25M-07	GREEN
GND	RJ45-05		RED
DTR	RJ45-06		BLACK
RTS	RJ45-07	DB25M-04	ORANGE
TX	RJ45-08	DB25M-02	BLUE

8917-04 WIRE LIST

FUNCTION	FROM	TO	COLOUR
RX	RJ45-01	DB25F-02	WHITE
CTS	RJ45-02	DB25F-04	BROWN
DSR	RJ45-03		YELLOW
GND	RJ45-04	DB25F-07	GREEN
GND	RJ45-05		RED
DTR	RJ45-06		BLACK
RTS	RJ45-07	DB25F-05	ORANGE
TX	RJ45-08	DB25F-03	BLUE

8917-03 WIRE LIST

FUNCTION	FROM	TO	COLOUR
RX	RJ45-01	DB9M-02	GREY
CTS	RJ45-02	DB9M08	BROWN
DSR	RJ45-03	DB9M-06	YELLOW
GND	RJ45-04	DB9M-05	GREEN
GND	RJ45-05		RED
DTR	RJ45-06	DB9M-04	BLACK
RTS	RJ45-07	DB9M-07	ORANGE
TX	RJ45-08	DB9M-03	BLUE

8917-02 WIRE LIST

FUNCTION	FROM	TO	COLOUR
Rx	RJ45-01	DB9F-03	WHITE
GND	RJ45-04	DB9F-05	GREEN
Tx	RJ45-08	DB9F-02	BLUE

[1] Not all ports shown are necessarily supported by the hardware. Consult the appropriate Instruction manual for details of the ports available on specific hardware.

4. Configuring the FieldServer as a Modbus RTU or Modbus ASCII Client.

For a detailed discussion on FieldServer configuration, please refer to the FieldServer Configuration Manual. The information that follows describes how to expand upon the factory defaults provided in the configuration files included with the FieldServer (See ".csv" sample files provided with the FieldServer).

This section documents and describes the parameters necessary for configuring the FieldServer to communicate with a Modbus RTU or Modbus ASCII Server.

4.1. Data Arrays/Descriptors

The configuration file tells the FieldServer about its interfaces, and the routing of data required. In order to enable the FieldServer for Modbus RTU or Modbus ASCII communications, the driver independent FieldServer buffers need to be declared in the "Data Arrays" section, the destination device addresses need to be declared in the "Client Side Nodes" section, and the data required from the Servers needs to be mapped in the "Client Side Map Descriptors" section. Details on how to do this can be found below.

Note that in the tables, * indicates an optional parameter, with the bold legal value being the default.

Section Title		
Data_Arrays		
Column Title	**Function**	**Legal Values**
Data_Array_Name	Provide name for Data Array	Up to 15 alphanumeric characters
Data_Array_Format	Provide data format. Each Data Array can only take on one format.	FLOAT, BIT, UInt16, SInt16, Packed_Bit, Byte, Packed_Byte, Swapped_Byte
Data_Array_Length	Number of Data Objects. Must be larger than the data storage area required by the map descriptors for the data being placed in this array.	1-10,000

Example

```
// Data Arrays
Data_Arrays
Data_Array_Name,    Data_Array_Format,    Data_Array_Length
DA_AI_01,           UInt16,               200
DA_AO_01,           UInt16,               200
DA_DI_01,           Bit,                  200
DA_DO_01,           Bit,                  200
```

4.2. Client Side Connection Descriptors

Section Title		
Connections		
Column Title	**Function**	**Legal Values**
Port	Specify which port the device is connected to the FieldServer	P1-P8, R1-R2[2]
Baud*	Specify baud rate	110 – 115200, standard baud rates only. **9600**
Parity*	Specify parity	Even, Odd, **None**
Data_Bits*	Specify data bits	7, **8**
Stop_Bits*	Specify stop bits	**1** (Vendor limitation)
Protocol	Specify protocol used	Modbus RTU
		Modbus _RTU
		Modbus ASCII
		MB_ASCII
Handshaking*	Handshaking is not supported	**None**
Poll Delay*	Time between internal polls	0-32000s, **0.05s**

Example

Change protocol to MB_ASCII to use Modbus ASCII protocol

```
// Client Side Connections

Connections
Port,   Baud,   Parity,   Data_Bits,   Stop_Bits,   Protocol,      Handshaking,   Poll_Delay
P8,     9600,   None,     8,           1,           Modbus _RTU,   None,          0.100s
```

[2] Not all ports shown are necessarily supported by the hardware. Consult the appropriate Instruction manual for details of the ports available on specific hardware.

4.3. Client Side Node Descriptors

Section Title		
Nodes		
Column Title	**Function**	**Legal Values**
Node_Name	Provide name for Node	Up to 32 alphanumeric characters
Node_ID	Modbus station address of physical Server Node	1-255
Protocol	Specify protocol used	Modbus RTU
		Modbus_RTU
		Modbus ASCII
		MB_ASCII
Port	Specify which port the device is connected to the FieldServer	P1-P8, R1-R2[2]
Node_Type[3]	Set to Block_Mode if Remote Server Node (RSN) only supports Write Multiple – FC16 & FC15, and does not support FC05 or FC06	Block_Mode
Address_Type[4]	Specify Register Mapping Model	ADU,PDU, **-, Modicon_5digit**

Example:

Change protocol to MB_ASCII to use Modbus ASCII protocol

```
//    Client Side Nodes
// For devices where 65536 addresses are available in each memory area.
Nodes
Node_Name,          Node_ID,    Protocol,     Port    Address_Type
Modbus device 1,    1,          Modbus_RTU,   P8      ADU
Modbus device 2,    2,          Modbus_RTU,   P8      PDU
// For devices where only 9999 registers are available in each memory area.
Nodes
Node_Name,          Node_ID,    Protocol,     Port
Modbus device 3,    3,          Modbus_RTU,   P8
```

[3] If this parameter is not specified the default function codes will be FC 05 (Single_Coil) and FC 06 (Single_Register). Refer to Appendix A.3 for more information.

[4] Optional for Modicon 5 digit devices

4.4. Client Side Map Descriptors

4.4.1. FieldServer Related Map Descriptor Parameters

Column Title	Function	Legal Values
Map_Descriptor_Name	Name of this Map Descriptor	Up to 32 alphanumeric characters
Data_Array_Name	Name of Data Array where data is to be stored in the FieldServer	One of the Data Array names from "Data Array" section above
Data_Array_Offset	Starting location in Data Array	0 to maximum specified in "Data Array" section above
Function	Function of Client Map Descriptor	RDBC, WRBC, WRBX

4.4.2. Driver Related Map Descriptor Parameters

Column Title	Function	Legal Values
Node_Name	Name of Node to fetch data from	One of the Node names specified in "Client Node Descriptor" above
Data_Type[5]	Specify memory area	Address_Type = ADU
		Coil, Discrete_Input, Input_Register, Holding_Register, Single_Coil, Single_Register
		Address_Type = PDU
		FC01, FC02, FC03, FC04, FC05, FC06, FC15, FC16
		Address_Type = Modicon_5digit
		- (Dash), Single_Register, Single_Coil
Address	Starting address of read block	Address_Type = ADU
		1-65536
		Address_Type = PDU
		0-65535
		Address_Type = Modicon_5digit
		40001, 30001, etc
Length	Length of Map Descriptor	1-125 (For Analog polls), 1-800 (For Binary polls).
Data_Array_Low_Scale*	Scaling zero in Data Array	Any signed 32 bit integer in the range: -2,147,483,648 to 2,147,483,647. **0**
Data_Array_High_Scale*	Scaling max in Data Array	Any signed 32 bit integer in the range: -2,147,483,648 to 2,147,483,647. **100**
Node_Low_Scale*	Scaling zero in Connected Node	Any signed 32 bit integer in the range: -2,147,483,648 to 2,147,483,647. **0**
Node_High_Scale*	Scaling max in Connected Node	Any signed 32 bit integer in the range: -2,147,483,648 to 2,147,483,647. **100**

4.4.3. Timing Parameters

Column Title	Function	Legal Values
Scan_Interval*	Rate at which data is polled	0-32000, **1**

[5] Optional only for Modicon_5digit addressing, and only if Single writes do not need to be forced

4.4.4. Map Descriptor Examples.

```
//   Client Side Map Descriptors
// Note: All three examples below are addressing the same Modbus registers.

// For Nodes where Address_Type is ADU
Map_Descriptors
Map_Descriptor_Name,  Data_Array_Name,  Data_Array_Offset,  Function,  Node_Name,          Data_Type,          Address,  Length,  Scan_Interval
CMD_AI_01,            DA_AI_01,         0,                  RDBC,      MODBUS DEVICE1,    Input_Register,     1,        20,      1.000s
CMD_AO_01,            DA_AO_01,         0,                  RDBC,      MODBUS DEVICE1,    Holding_Register,   1,        20,      1.000s
CMD_DI_01,            DA_DI_01,         0,                  RDBC,      MODBUS DEVICE1,    Discrete_Input,     1,        20,      1.000s
CMD_DO_01,            DA_DO_01,         0,                  RDBC,      MODBUS DEVICE1,    Coil,               1,        20,      1.000s

// For Nodes where Address_Type is PDU
Map_Descriptors
Map_Descriptor_Name,  Data_Array_Name,  Data_Array_Offset,  Function,  Node_Name,          Data_Type,          Address,  Length,  Scan_Interval
CMD_AI_02,            DA_AI_02,         0,                  RDBC,      MODBUS DEVICE2,    FC04,               0,        20,      1.000s
CMD_AO_02,            DA_AO_02,         0,                  RDBC,      MODBUS DEVICE2,    FC03,               0,        20,      1.000s
CMD_DI_02,            DA_DI_02,         0,                  RDBC,      MODBUS DEVICE2,    FC02,               0,        20,      1.000s
CMD_DO_02,            DA_DO_02,         0,                  RDBC,      MODBUS DEVICE2,    FC01,               0         20,      1.000s

// For Nodes where Address_Type is Modicon_5digit.
Map_Descriptors
Map_Descriptor_Name,  Data_Array_Name,  Data_Array_Offset,  Function,  Node_Name,                                  Address,  Length,  Scan_Interval
CMD_AI_03,            DA_AI_03,         0,                  RDBC,      MODBUS DEVICE3,                            30001,    20,      1.000s
CMD_AO_03,            DA_AO_03,         0,                  RDBC,      MODBUS DEVICE3,                            40001,    20,      1.000s
CMD_DI_03,            DA_DI_03,         0,                  RDBC,      MODBUS DEVICE3,                            10001,    20,      1.000s
CMD_DO_03,            DA_DO_03,         0,                  RDBC,      MODBUS DEVICE3,                            00001,    20,      1.000s
```

5. Configuring the FieldServer as a Modbus RTU or Modbus ASCII Server

For a detailed discussion on FieldServer configuration, please refer to the FieldServer Configuration Manual. The information that follows describes how to expand upon the factory defaults provided in the configuration files included with the FieldServer (See ".csv" sample files provided with the FieldServer).

This section documents and describes the parameters necessary for configuring the FieldServer to communicate with a Modbus RTU or Modbus ASCII Client.

The configuration file tells the FieldServer about its interfaces, and the routing of data required. In order to enable the FieldServer for Modbus RTU or Modbus ASCII communications, the driver independent FieldServer buffers need to be declared in the "Data Arrays" section, the FieldServer virtual Node(s) needs to be declared in the "Server Side Nodes" section, and the data to be provided to the clients needs to be mapped in the "Server Side Map Descriptors" section. Details on how to do this can be found below.

Note that in the tables, * indicates an optional parameter, with the **bold** legal value being the default.

5.1. Server Side Connection Descriptors

Section Title		
Connections		
Column Title	**Function**	**Legal Values**
Port	Specify which port the device is connected to the FieldServer	P1-P8, R1-R2[6]
Baud*	Specify baud rate	110 – 115200, standard baud rates only
Parity*	Specify parity	Even, Odd, None
Data_Bits*	Specify data bits	7, **8**
Stop_Bits*	Specify stop bits	**1** (Vendor limitation)
Protocol	Specify protocol used	Modbus RTU
		Modbus_RTU
		Modbus ASCII
		MB_ASCII
Handshaking*	Handshaking is not supported.	**None**

Example

Change protocol to MB_ASCII to use Modbus ASCII protocol

```
// Server Side Connections

Connections
Port,       Baud,    Parity,  Data_Bits,  Stop_Bits,  Protocol,    Handshaking
P1,         9600,    None,    8,          1,          Modbus_RTU,  None
```

[6] Not all ports shown are necessarily supported by the hardware. Consult the appropriate Instruction manual for details of the ports available on specific hardware.

5.2. Server Side Node Descriptors

Section Title		
Nodes		
Column Title	**Function**	**Legal Values**
Node_Name	Provide name for Node	Up to 32 alphanumeric characters
Node_ID	Node ID of physical Server Node	1 – 255
Protocol	Specify protocol used	Modbus RTU
Address_Type[7]	Specify Register Mapping Model	ADU,PDU, -, **Modicon_5digit**

Example

Change protocol to MB_ASCII to use Modbus ASCII protocol

```
//    Server Side Nodes
// For devices where 65536 addresses are available in each memory area.
Nodes
Node_Name,          Node_ID,    Protocol        Address_Type
MB_Srv_11,          11,         Modbus_RTU      ADU
MB_Srv_12,          12,         Modbus_RTU      PDU
// For devices where only 9999 registers are available in each memory area.
Nodes
MB_Srv_13,          13,         Modbus_RTU      Modicon_5digit
MB_Srv_14,          14,         Modbus_RTU      -
```

[7] Optional for Modicon 5 digit devices

5.3. Server Side Map Descriptors

5.3.1. FieldServer Specific Map Descriptor Parameters

Column Title	Function	Legal Values
Map_Descriptor_Name	Name of this Map Descriptor	Up to 32 alphanumeric characters
Data_Array_Name	Name of Data Array where data is to be stored in the FieldServer	One of the Data Array names from "Data Array" section above
Data_Array_Offset	Starting location in Data Array	0 to maximum specified in "Data Array" section above
Function	Function of Server Map Descriptor	Server

5.3.2. Driver Specific Map Descriptor Parameters

Column Title	Function	Legal Values
Node_Name	Name of Node to fetch data from	One of the Node names specified in "Client Node Descriptor" above
Data_Type[8]	Specify memory area	Address_Type = ADU
		Coil, Discrete_Input, Input_Register, Holding_Register, Single_Coil, Single_Register
		Address_Type = PDU
		FC01, FC02, FC03, FC04, FC05, FC06, FC15, FC16
		Address_Type = Modicon_5digit
		- (Dash), Single_Register, Single_Coil
Length	Length of Map Descriptor	1-10000.
Address	Starting address of read block	Address_Type = ADU
		1-65536
		Address_Type = PDU
		0-65535
		Address_Type = Modicon_5digit
		40001, 30001, etc
Data_Array_Low_Scale*	Scaling zero in Data Array	Any signed 32 bit integer in the range: -2,147,483,648 to 2,147,483,647. **0**
Data_Array_High_Scale*	Scaling max in Data Array	Any signed 32 bit integer in the range: -2,147,483,648 to 2,147,483,647. **100**
Node_Low_Scale*	Scaling zero in Connected Node	Any signed 32 bit integer in the range: -2,147,483,648 to 2,147,483,647. **0**
Node_High_Scale*	Scaling max in Connected Node	Any signed 32 bit integer in the range: -2,147,483,648 to 2,147,483,647. **100**
Node_Offline_Response	Set the FieldServer response to the Modbus RTU Client when the Server Node supplying the data has gone offline	No_Response, Old_Data, Zero_Data, FFFF_Data, Refer to Appendix A.6 for further information.

[8] Optional only for Modicon_5digit addressing, and only if Single writes do not need to be forced

5.3.3. Map Descriptor Examples

```
//   Server Side Map Descriptors where Node Address_Type is ADU
// Note: All three examples below are addressing the same Modbus registers.

// For Nodes where Address_Type is ADU
Map_Descriptors
Map_Descriptor_Name   Data_Array_Name   Data_Array_Offset,   Function   Node_name,   Data_type          Address,   Length,   Data_Array_Low_Scale,   Data_Array_High_Scale   Node_Low_Scale   Node_High_Scale
SMD_AI_01,            DA_AI_01,         0,                   Server,    MB_Srv_11    Input_Register     1,         200,      0,                     100,                    0,               10000
SMD_AO_01,            DA_AO_01,         0,                   Server,    MB_srv_11    Holding_Register   1,         200,      0,                     100,                    0,               10000
```

```
//   Server Side Map Descriptors where Node Address_Type is PDU
Map_Descriptors
Map_Descriptor_Name   Data_Array_Name   Data_Array_Offset,   Function   Node_name,   Data_type   Address,   Length,   Data_Array_Low_Scale,   Data_Array_High_Scale   Node_Low_Scale   Node_High_Scale
SMD_AI_02,            DA_AI_02,         0,                   Server,    MB_Srv_12    FC04        0,         200,      0,                     100,                    0,               10000
SMD_AO_02,            DA_AO_02,         0,                   Server,    MB_srv_12    FC03        0,         200,      0,                     100,                    0,               10000
```

```
// For Nodes where Address_Type is Modicon_5digit.
Map_Descriptors
Map_Descriptor_Name,   Data_Array_Name,   Data_Array_Offset,   Function,   Node_name,   Address,   Length,   Data_Array_Low_Scale,   Data_Array_High_Scale   Node_Low_Scale   Node_High_Scale
SMD_AI_01,             DA_AI_01,          0,                   Server,     MBP_Srv_13,   30001,     200,      0,                     100,                    0,               10000
SMD_AO_01,             DA_AO_01,          0,                   Server,     MBP_Srv_13,   40001,     200,      0,                     100,                    0,               10000
```

Appendix A. Advanced Topics – Modbus RTU

Appendix A.1. Data Types

If Node parameter Address_Type is set as ADU or PDU, then Data_Type must be specified as follows

For Address_Type ADU :

Address range	Data_Type	Function Code (Write)	Function Code (Read)
1 - 65536	Coil	15	1
1 – 65536	Discrete_Input	n/a.	2
1 – 65536	Input_Register	n/a.	4
1 - 65536	Holding_Register	16	3

For Address_Type PDU :

Address range	Data_Type	Function Code (Write)	Function Code (Read)
0 - 65535	FC01	15	1
0 – 65535	FC02	n/a.	2
0 – 65535	FC04	n/a.	4
0 – 65535	FC03	16	3

For Address_Type Modicon_5digit

When a Modbus address range is specified, a particular Data Type is implied. The defaults are as follows:

Address range	Data_Type	Function Code (Write)	Function Code (Read)
40001 - 49999	Register	16	3
30001 - 39999	Analog _Input	n/a.	4
10001 - 19999	Digital_Input	n/a.	2
00001 - 09999	Coil	15	1

Appendix A.2. Single Writes

For pure write operations where the function = WRBC or WRBX, the driver defaults to using Function Codes 15 and 16 (Multiple writes). It is possible to force the driver to use Function Codes 5 and 6 (Single Writes) by manipulating the Data_Type parameter as follows:

For Address_Type ADU:

Address range	Data_Type	Function Code (Write)
1 - 65536	Single_Coil	5
1 - 65536	Single_Register I	6

For Address_Type PDU:

Address range	Data_Type	Function Code (Write)
0 - 65535	FC05	5
0 - 65535	FC06	6

For Address_Type Modicon_5digit

Address range	Data_Type	Function Code (Write)
40001 - 49999	Single_Register	6
30001 - 39999	Single_Coil	5.

Example: FC 6 = Write Single Register

Add a parameter to the Modbus client side Map Descriptor called Data_Type.
If you specify the Data_Type as Single_Register and the Function as WRBC or WRBX, then a Modbus poll with FC 6 will be generated.
Logically Single Register implies a length of one, and even if you try to set the length longer in the csv file, the length is limited to 1 in the driver.

Appendix A.3. Read/write Operation

When using the driver as a Modbus master, the function RDBC allows read/write capability with Register and Coil data types. If defaults are used, then Function codes 5 and 6 (Single Writes) are used to write data back to the registers being read, regardless of data length being read. If multiple writes (FC 15 and 16) are needed for Read/write operation, the user needs to specify the Node_Type parameter in the Client Side Nodes Section and set it to Block_Mode.

Note that block writes of length 1 are currently all that is supported.

Appendix A.4. Managing Floating points with Modbus

Modbus as a standard does not support floating point formats. Many vendors have written higher level communications software to use two 16 bit registers to represent floating point or 32 bit integers. This requires conversion software on both ends of the communication channel. The FieldServer supports this function and also provides other options to resolve this issue.

Transferring non-integer values with one register

It is possible to represent values higher than 32767 using one register in one of two ways:

- Declare data arrays as type Uint16 (Unsigned integer). This will give you a range from 0 to 65535.
- Use the scaling function on the FieldServer, which will allow you to set up any range, with 16 bit resolution.

The following example shows how scaling can be achieved on the Server side of the configuration. Note that scaling can also be done on the Client side to scale down a value that was scaled up by a Modbus vendor. Further information regarding scaling can be found in the FieldServer Configuration manual.

Example :

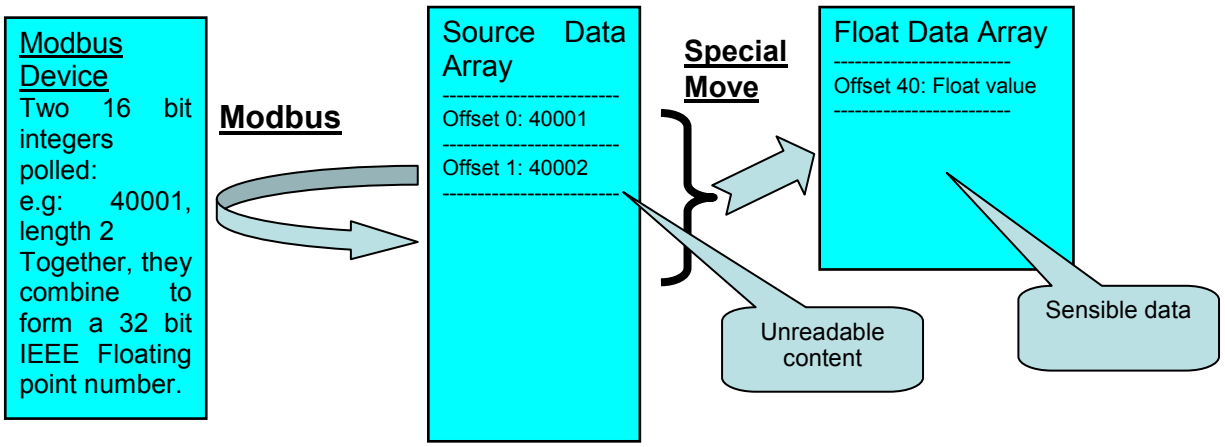
```
Map_Descriptors
Map_Descriptor_Name,  Data_Array_Name,  Data_Array_Offset,  Function,  Node_Name,  Address,  Length,  Data_Array_Low_Scale,  Data_Array_High_Scale,  Node_Low_Scale,  Node_High_Scale,
SMD_AI1,              DA_AI_01,         0,                  Server,    MBP_Srv_11,  30001,    200,     0,                  100,                   0,              10000
SMD_AO1,              DA_AO_01,         0,                  Server,    MBP_Srv_11,  40001,    200,     0,                  100,                   0,              10000
```

This example multiplies the values in the data array by 100 (10000 on Node_High_Scale is 100X larger than 100 on Data_Array_High_Scale). This is most commonly used when the user wants to introduce values after the decimal point. For example, a value of 75.6 will be sent as 7560, which can then be rescaled by the Modbus master.

Transferring 32 bit values with two registers

If a Modbus Server sends two consecutive registers to the FieldServer representing either a floating point value or a 32 bit integer value, the FieldServer can combine and decode these registers back into their original format. To do this, the user must read the registers into an integer data array, and then use the Moves function to move the data to a floating point or 32 bit integer data array. The "function" field in the move must be populated with the correct move function required. Details of the available function fields are listed in the FieldServer configuration manual along with a more detailed explanation of moves. Note that functions also exist to split a floating point value into two integers using the reverse operation.

A diagrammatic representation of this process is depicted below:



The example below shows how a floating point value is retrieved from two Modbus registers that were placed in an integer data array.

Example

```
Data_Arrays
Data_Array_Name,    Data_Format,    Data_Array_Length
DA1,                int16,          20
DA2,                float,          10
```

```
Moves
Function,           Source_Data_Array,  Source_Offset,  Target_Data_Array,  Target_Offset
2.i16-1.float-sw,   DA1,                0,              DA2,                6
```

This Move will cause two integers at offsets 0 and 1 in the integer Data Array DA1 to be copied to offset 6 in the float Data Array DA2. In the process of copying the values, the integers will be combined and treated as an IEEE 32 bit floating point number.

Appendix A.5. Connection to York Modbus Microgateway

If connecting the FieldServer to a York Modbus Microgateway, the Node_ID of the Microgateway is defined by the address DIP switches. If switch 4 is set to 'On' and the other switches are set to 'off' then Node_ID of the Microgateway is '247', the parity is 'Even', and the stop bits are 1. Other Node_ID combinations can be found in the York Modbus Microgateway Installation Manual.

Appendix A.6. Node_Offline_Response

This function is specific to the Modbus RTU driver.

In systems where data is being collected from multiple Server Nodes and made available on a FieldServer configured as a Modbus RTU Server, when a Server Node goes offline the default behavior of the FieldServer would be to stop responding to polls for this data. This might not be what the user wants. Various options exist making it possible to signal that the data quality has gone bad without creating error conditions in systems sensitive to the default option.

The following options can be configured under the Node parameter, Node_Offline_Response, to set the response of the FieldServer to the Modbus RTU Client when the Server Node supplying the data is offline:

- No_Response - this is the default option. The FieldServer simply does not respond when the corresponding Server Node is offline.
- Old_Data - The FieldServer will respond, but with the last known value of the data. This maintains the communication link in an active state, but may hide the fact that the Server Node is offline.
- Zero_Data - The FieldServer will respond, but with the data values set to zero. If the user normally expects non-zero values, this option will signal the offline condition without disrupting communications.
- FFFF_Data - The FieldServer will respond, but with the data values set to FFFF (hex). If the user normally expects other values, this option will signal the offline condition without disrupting communications.

Example:

Nodes				
Node_name,	Node_ID,	Protocol,	Node_Offline_Response,	port
DEV11,	11,	Modbus_rtu,	No_Response,	-
DEV12,	12,	Modbus_rtu,	Old_Data,	-
DEV15,	15,	Modbus_rtu,	Zero_Data,	-
DEV16,	16,	Modbus_rtu,	FFFF_Data,	-

Appendix B. Modbus ASCII - Examples of FieldServer setup for typical clients

Appendix B.1. FieldServer with GE Cimplicity as client

- Run the Cimplicity "Workbench" and create a "New Project" with a unique "Project Name" option of "Basic Control" and protocol "Modbus ASCII".
- Check the project properties and continue with the "Project Wizard Setup" that appears.
- Add Modbus port giving it a description.
- Create and configure the devices, select "new item".
- Name the device, select the port, give it a description (e.g. FieldServer), and choose "SYSTEM" resource.
- Create and configure the points.
- Select "new item", name the point and choose the appropriate device.
- Under the "General" tab, point properties require a description. Note that the elements must have a value greater than 8.
- Under the "Device" tab, properties need the appropriate address (e.g. 40001 and also require the leading 0's), change the update criteria to "On Scan".
- When the project is configured, run by pressing the "play" button.
- Expect the Cimplicity driver to connect and poll the FieldServer for a range of valid addresses, and then proceed to poll for just the configured Points.
- From the start menu choose the "Point Control Panel", select edit and add the project you want to view. Note, to log on the User name "ADMINISTRATOR" must be supplied
- Use "Modbus ASCII Diagnostics" to connect to host and then read the register.

Appendix B.2. FieldServer with Intellution FIX as a client

- Install Intellution FIX, choosing the MB1 Modbus ASCII I/O driver
- Run from Start menu and choose "Intellution Fix".
- Choose "System Configuration Utility".
- Modify SCADA, add the MB1 Modbus ASCII I/O driver
- Configure the Modbus ASCII Driver.
- Device is D11, select 5-digit address, add the FieldServer virtual Node ID to station address
- Set up poll record
- SAVE the configuration.
- Open "StaRTUp"
- Open "Mission Control" from the "Apps" menu and confirm Fix is polling.
- To display the data create a link in Fix draw, add link, data link.
- Give it a tagname, allow data entry, numeric entry and set enable option.
- If tag is not in database, select "Add", choose "AR". Then set output enable , device MB1, I/O address d11.
- Save the settings
- Use "Quickview" from the "View" menu to confirm the reading of data without ??? appearing
- Change the value and wait a few seconds to ensure the change really occurred.

THIS PAGE INTENTIONALLY LEFT BLANK